REM Sleep Monitor II 2013

User's Manual

Alex Jenkins

Britta Olson

James Parrow

Ryan Klemisch

Table of Contents

INTRODUCTION:

The REM sleep monitor detects REM sleep using the electrooculogram (EOG). The EOG is a potential between the back and the front of the human eye. When the eye looks left and then right, there is a change in the polarity between the retina and the cornea. The EOG is very small, generally only a couple micro volts. This device obtains a signal by wearing a headband with built in electrodes. The device reads the EOG signal and detects whether or not the user is in REM sleep. The device alerts the user via a speaker subtly once they've entered REM sleep and directly after exiting REM an irksome alert is heard. The intent of the device is to induce lucid dreaming, when the user is aware that they are dreaming. This type of device has the potential to help better understand the science of dreaming, specifically lucid dreaming. Currently, REM sleep devices are very expensive; the goal is to design a cost effective device that is able to be replicated by the general public. A "How to Guide" is now available for the public to make a proto-board at home. The proto-board is to be populated on a breadboard. An official PCB version of the device was fabricated as well. The PCB is shown below:

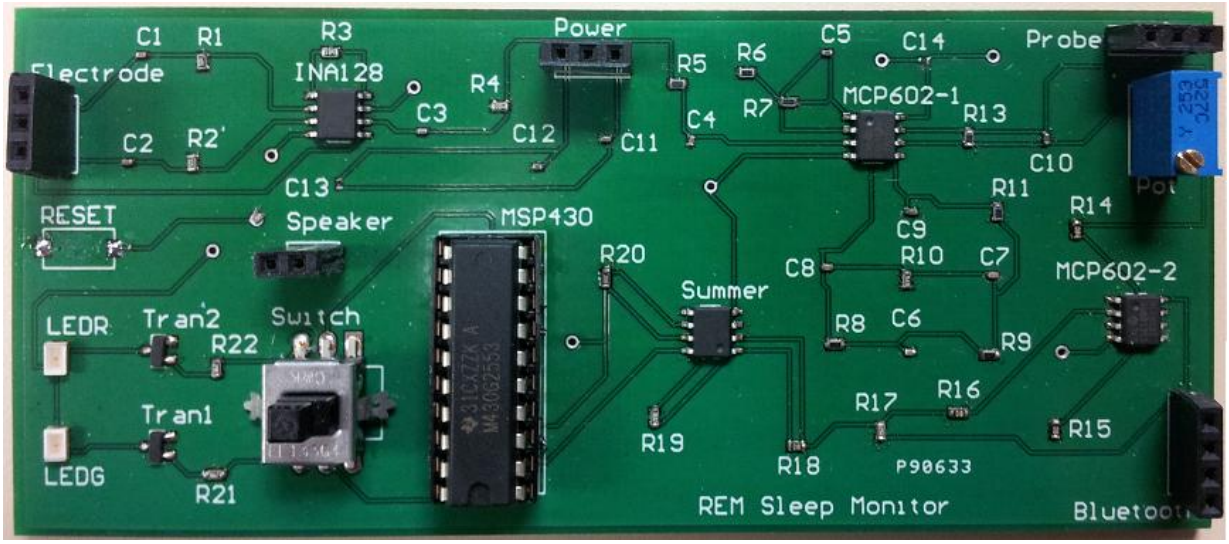


Figure1: Printed Circuit Board (PCB)

INPUTS:

Power

The Power header has three pins: +3V, -3V, and GND. All pins must be connected for the device to operate.

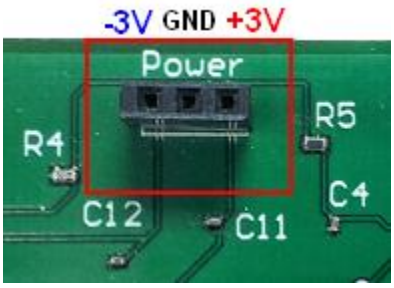


Figure2: Power Header

Headband

The headband should be worn tightly around the head so the electrodes touch the forehead. The middle (reference) electrode should be centered between the eyes. The outer electrodes (+/-) should be placed above and towards the outer corner of each eye. The electrode placement is shown below:

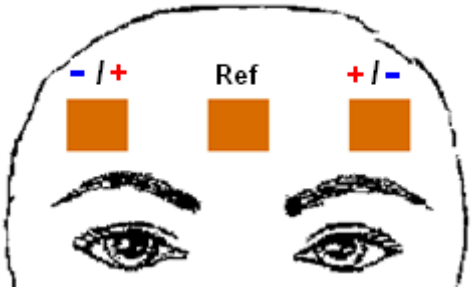


Figure3: Electrode Placement

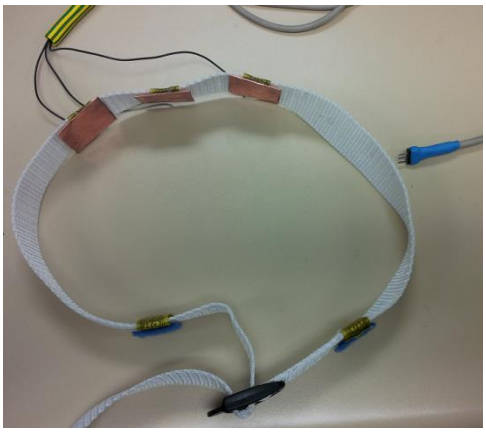


Figure4: Headband

Electrodes

The electrode header has three pins: +, -, and Reference. The top and bottom pins may be interchanged, however, the middle pin must always be reference. All pins must be connected for the device to operate.

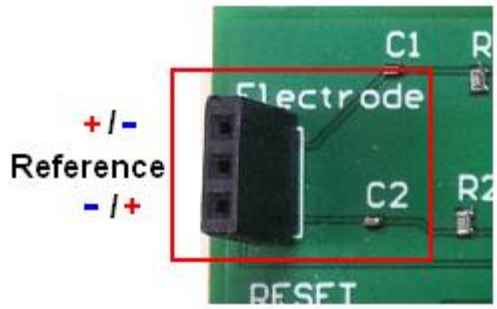


Figure5: Electrode Header

Potentiometer

The potentiometer allows the user to adjust the gain of their signal. If the user's eye movements are not large enough to cross the REM threshold, the gain must be increased. If the user's eye movements saturate, then the gain must be decreased. Turning the potentiometer clockwise increases the gain, and turning the potentiometer counterclockwise decreases the gain.

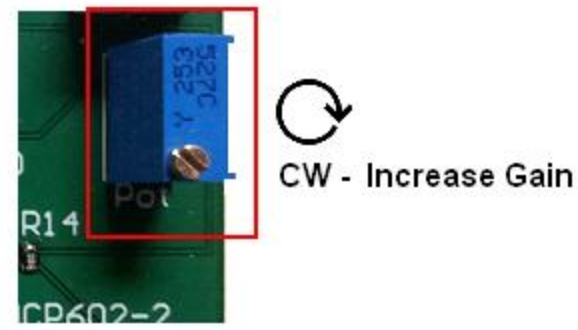


Figure6: Potentiometer

SWITCH

The switch turns the indication LEDs ON and OFF. Pushing the switch once will activate the LED function. Similarly, pushing the switch a second time will deactivate the LED function.

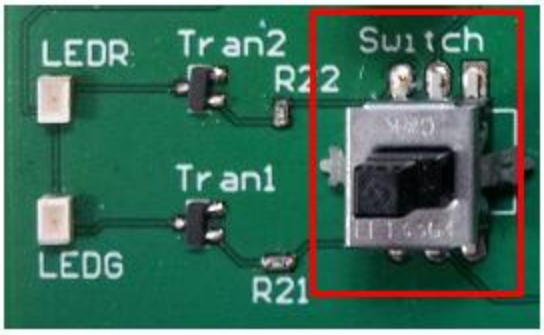


Figure7: LED Switch

Microprocessor

The MSP430 microprocessor is used for this device. The microprocessor can be programmed to the user's preferences (See page 14). The microprocessor must be placed in the board for the device to operate.

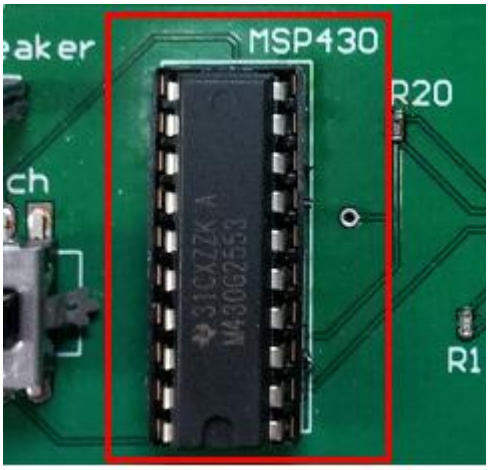


Figure8: MSP430 Microprocessor

OUTPUTS:

Speaker

The speaker header has two pins. The polarity of the speaker does not matter, so either lead from the speaker can be plugged into either pin. Both pins must be connected for the speaker to operate. However, the rest of the device will operate properly without the speaker connected.

The speaker has two modes:

- Low frequency (250Hz)
- High frequency (2500Hz)

The low frequency mode is sounded when the user enters REM. This soft noise allows the user to recognize the sound while dreaming in the attempts to lucid dream. When the user exits REM the harsh high frequency noise will sound to wake the user to help remember the dream experience.

Figure9: Speaker

Bluetooth

The Bluetooth header has four pins: Vcc, GND, TXD, and RXD. The corresponding pins on the Bluetooth Module JY-MCU plug into the header on the PCB board. All pins must be connected for the Bluetooth Module to transmit. However, the rest of the device will operate properly without the Bluetooth Module connected.

The Bluetooth Module transmits the EOG signal from the PCB to a PC with a Bluetooth receiver and LabVIEW. LabVIEW displays the EOG signal, and allows the user to log their sleep data. (See page 10)

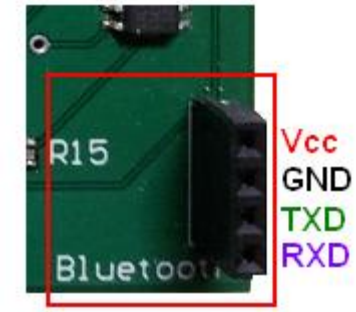


Figure10: Bluetooth Header

Figure11: JY-MCU Bluetooth Module

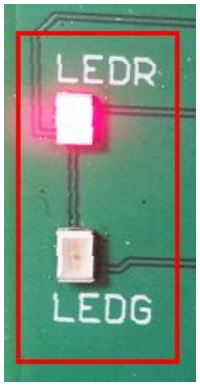
Probe

The probe points allow the user to view the signal at different points in the circuit if they have access to an oscilloscope. Probe point P1 reads the signal right before it is fed into the microprocessor (0-3V). Probe point P2 reads the signal before the summing circuit (-3 - 3V). In order to read a probe point, one oscilloscope lead must always be grounded. To read P1, one lead should be in pin P1, and the other should be in GND. To read P2, one lead should be in pin P2, and the other should be in GND. The device will operate normally with or without the probe header connected.

Figure12: Probe Points Header

LEDs

There are two indication LEDS on the PCB. The top LED (RED) indicates that the user's eye movements are not in the REM threshold. The bottom LED (GREEN) indicates that the user's eye movements are in the REM threshold. One LED will always be illuminated.



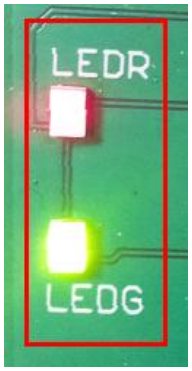


Figure13: Red LED **Figure14**: Green L

Figure15: PCB Enclosure

LabVIEW:

LabVIEW allows the user to view and log their sleep data via Bluetooth. There is a REM indicator, which illuminates if the user's EOG signal is in the REM threshold. LabVIEW also displays a chart, which graphs the EOG signal in real time. Data automatically saves to a file when REM.vi is started.

When the user opens the REM.vi in LabVIEW the front panel will appear. The front panel is the user interface, which allows the user to run and stop the vi. Below, the front panel is displayed:

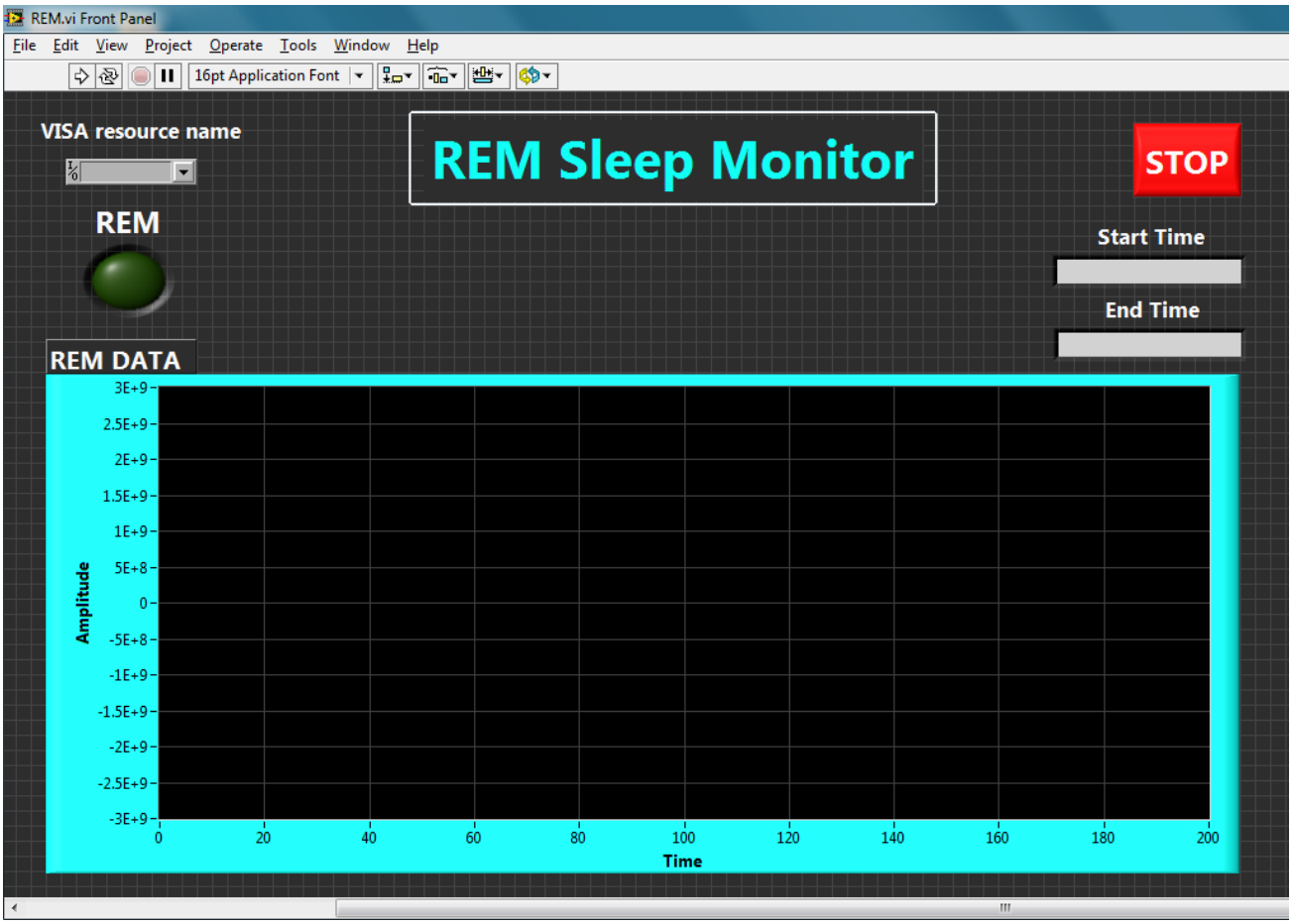


Figure16: LabVIEW Front Panel

By clicking Window => Show Block Diagram, the user can view the block diagram. The block diagram is where the program is created, and allows the user to edit the program. Below, the Block Diagram is displayed:

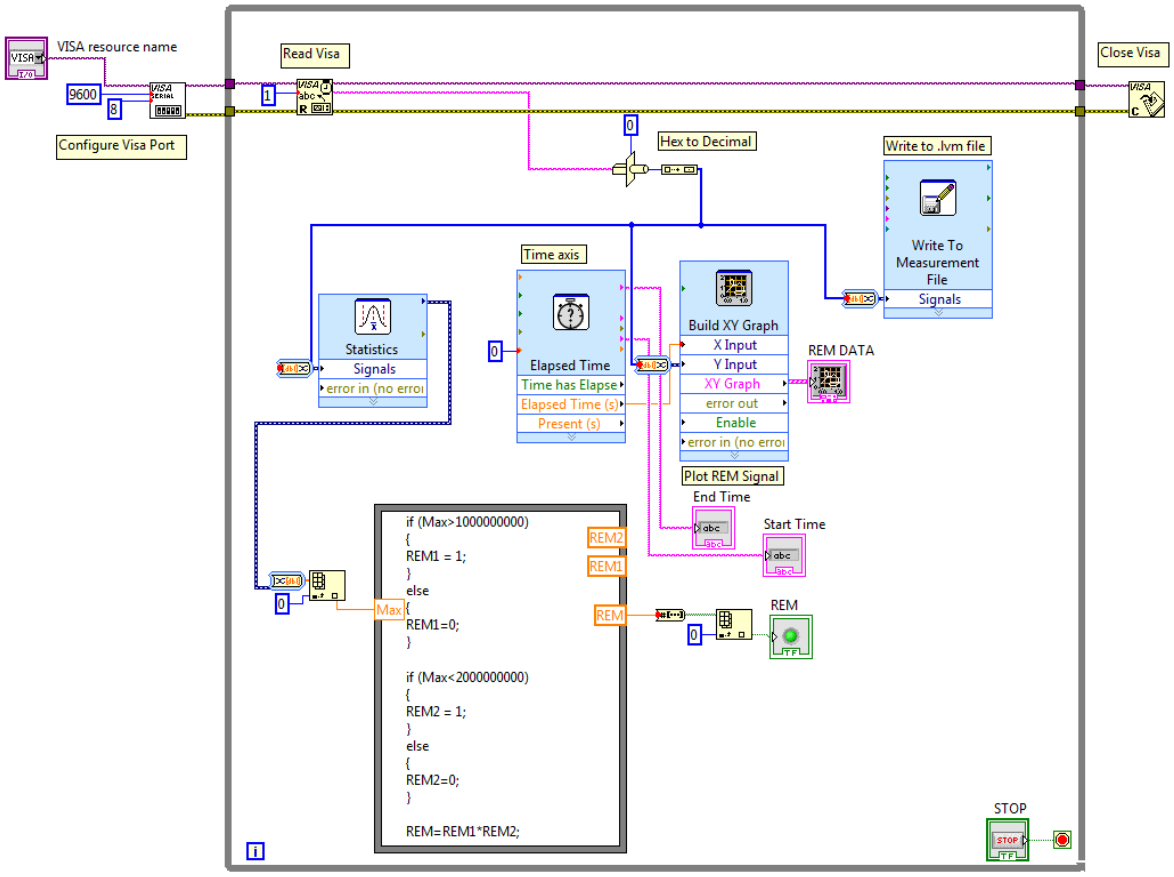


Figure17: LabVIEW Block Diagram

Setting up Bluetooth:

The JY-MCU Bluetooth Module transmits data to any generic USB Bluetooth receiver(or PC with internal Bluetooth). The USB Bluetooth receiver must be plugged into a USB port on a PC. The front panel of the LabVIEW program allows the user to select which COM Port the USB Bluetooth receiver is located in.

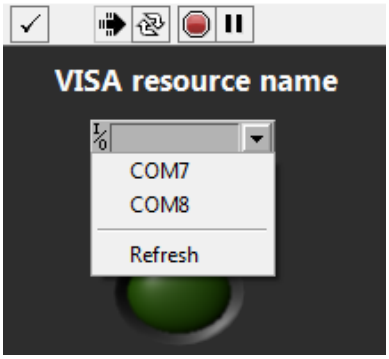


Figure18: VISA Resource

Logging Data:

The LabVIEW program creates a .lvm data file when the program is run. Every time the program is run, it will save over the previous data file. This means that if the user wants to save their data, they must rename the file. To change the location the file is saved to the user must:

- Double Click the Write To Measurement vi on the Block Diagram

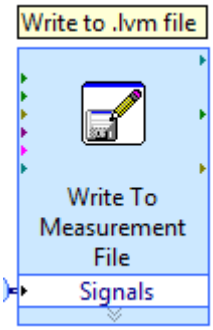


Figure19: Write to Measurement vi

- Insert the desired location in space below **Filename**, and press **OK**

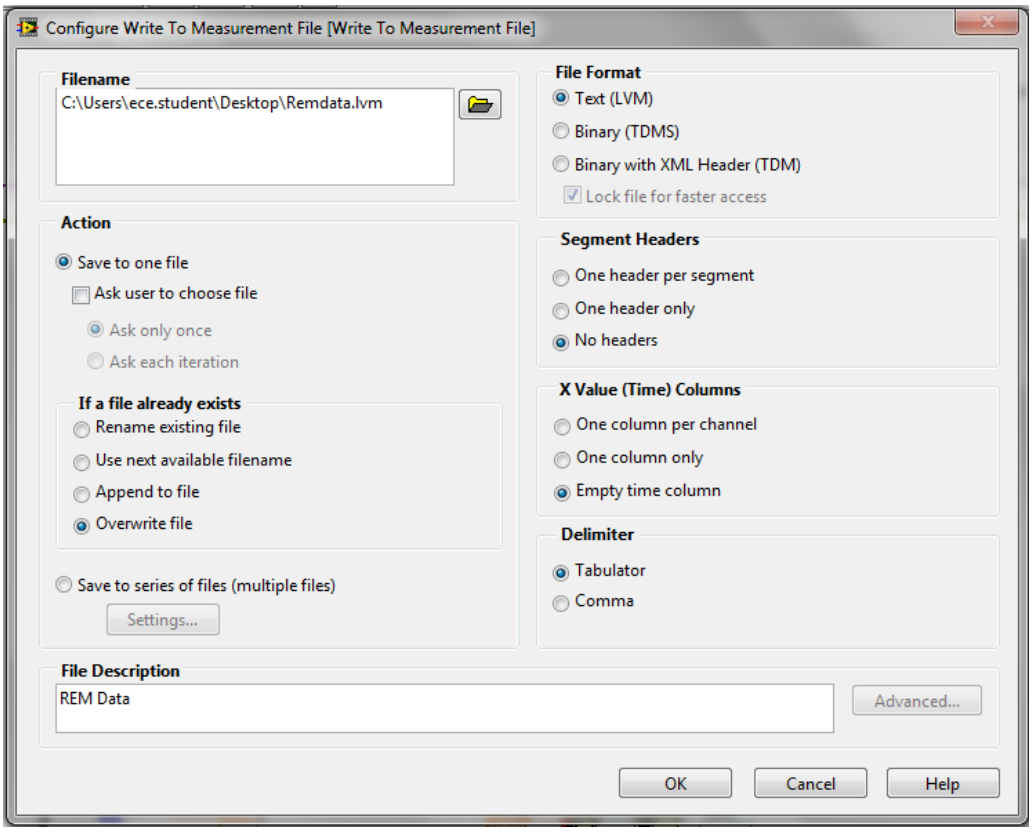


Figure20: Assigning data location

Running LabVIEW:

To start the program, the user must click the white arrow located on the top bar of LabVIEW. To stop the program the user may either click the stop sign located on the top bar of LabVIEW, or the STOP button located on the front panel.

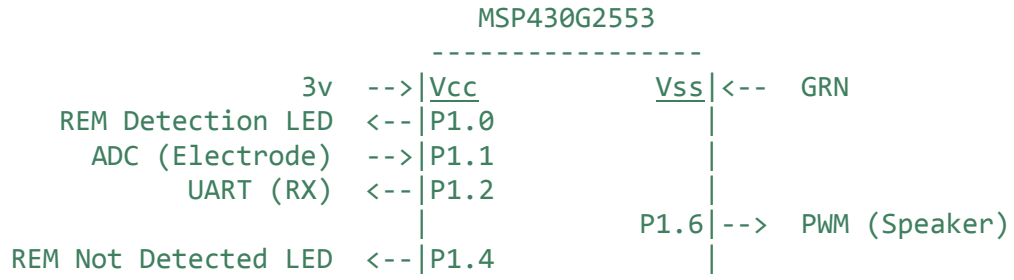
Figure21: Start Button

Figure22: Stop Buttons

Software:

MSP430 Microprocessor

This section will go into detail about some features a user can change by programming the MSP430G2553.



Initialize Variables

Initializing the following variables will allow the user to set up the REM Monitor to their own personal ideal settings.

❖ `int ScanPeriodInREM = 457;  // 457 is approx 30 seconds`

This variable will allow the user to program a scan period for REM. The scan period acts as a checkpoint, if REM is positive during this scan period the checkpoint is successful. Note when setting ScanPeriodInREM, if the period is too long then detecting REM may not be accurate. For example if ScanPeriodInREM is 5 minutes, it is possible that the user was in REM for the first 5 seconds of the 5 minute scan period, this would result in a successfully checkpoint even though the individual may have come out of REM sleep. Another example, if ScanPeriodInREM is 0.2 seconds, the user could be in REM but in order to turn on the Low Frequency speaker a desired NumberOfScanPeriods needs to be met in a row. Choosing 30 seconds maybe a good medium.

To solve for ScanPeriodInREM:

$$\text{"Seconds in ScanPeriodInREM"} = (ScanPeriodInREM) * 0.065536$$

Note that 0.065536 comes from the clock settings and how many bits the MSP430G3553 is. The MSP430G2553 is 16 bits and the clock is set for 1Mhz.

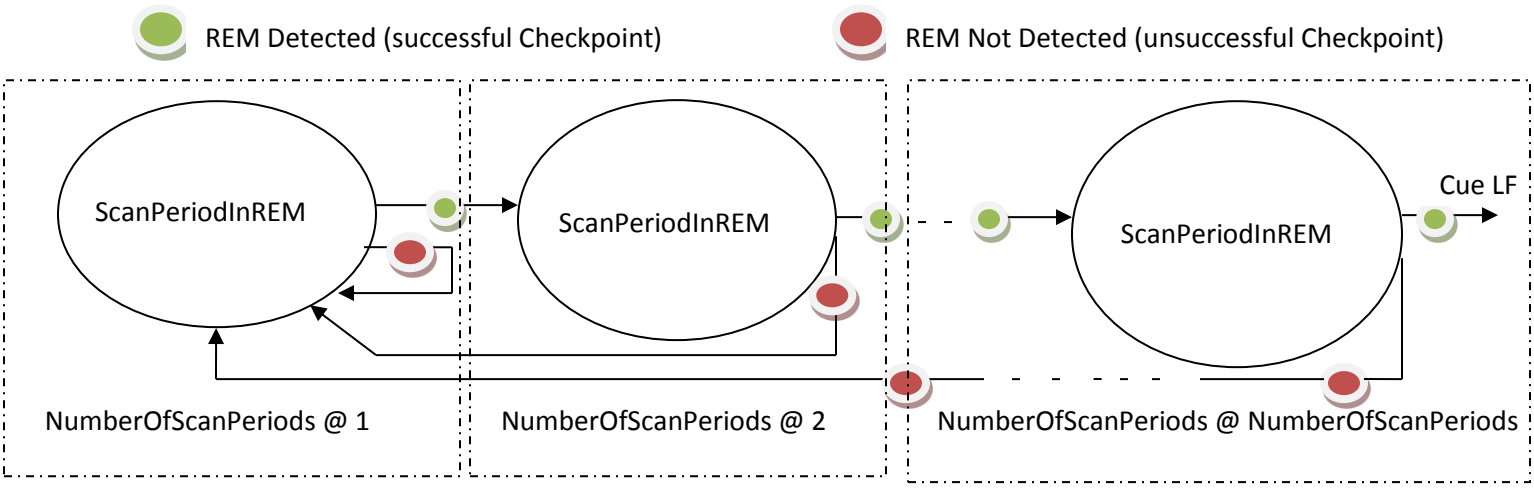
$$\text{"Time per Clock cycle"} = \frac{2^{16}}{1\text{Mhz}} = \frac{65536}{1000000} = 0.065536$$

❖ `int NumberOfScanPeriods = 10;  // 10 periods is approx 5 mins total time scanning`

This variable will decide how many checkpoints (ScanPeriodInREM) are needed to cue the user if they are lucid dreaming. Setting this variable will choose how long an individual will be in REM before cueing; this variable is affected by ScanPeriodInREM as shown below.

$$\text{"Time in REM"} = (NumberOfScanPeriods) * (ScanPeriodInREM)$$

In the code set up ScanPeriodInREM = 457 and NumberOfScanPeriods = 10. So this means that approximately 300 seconds are needed in REM before the speaker is turned on at a low frequency. Note that if a checkpoint is unsuccessful then NumberOfScanPeriods will go back to zero. If enough data is gathered for the individual, that data could be used to set ScanPeriodInREM and NumberOfScanPeriods which would increase the accuracy of detecting REM.



- `int MinsInLowFre = 5;  // 5 is approx 5 mins that Low Frequency speaker is sounded (this is the lucid dream cue)`

This variable will determine how long the speaker is sounded at a low frequency. The number that MinsInLowFre equals will be how many minutes that the speaker is sounding at low frequency before it sounds in high frequency. Low frequency is used to cue the user that they are dreaming. Note in the code example, REM was scanned for 5 minutes then low frequency was sounded for 5 minutes, making a total of 10 minutes that an individual should be in REM. The goal is to have the speaker sound at a high frequency to wake the user directly after REM. With this goal, the user would be able to remember the lucid dream experience.

Registers

The following statements are used to set up the MSP430G2553. All of the registers can be found in the MSP430G2553 data sheet. The following section will going into more detail about the more complicated registers.

- `TACTL = TASSEL_2 + MC_2;  // Set the timer A to SMCLCK, Continuous`

This statement sets up the clock at 1Mhz, changing this would result in different times for all statements. Note that this statement is used later on to change the clock speed for pulse wave modulating.

- `ADC10CTL1 = INCH_1; // input A1`

This statement sets up the ADC (Analog to Digital Converter) reference voltages. INCH_1 uses the battery as reference voltage, making whatever voltage is on VCC (up to 3.6v) is the high reference and whatever voltage is on GND (down to -2 volts) is the lower reference. Note that the maximum potential difference between VCC and GND can only be 4volts. The MSP430G2553 has internal references that can be used, one at 2.5v, but we chose to use the battery as a reference voltage to give us a wider range. Changing this will affect the ADC values.

Main

Below is the main code, this section will go more into detail about the algorithm.

- `_BIS_SR(LPM0_bits + GIE);  // Enter LPM3 // Exit is located in interrupt, not efficient with no crystal and ADC interrupt`

The statement above enters the MSP430G2553 into low power mode. The goal of this is to save battery life while the user is still awake and attempting to fall asleep. Low power mode can be exited after a chosen period of time found in the timer interrupt service routine. This is a feature that can be implemented if desired. LPM is not used for now for a few reasons: using the LPM3 without a crystal is said to be inefficient, the interrupt routines may make LPM inefficient. Until LPM is proven to save energy in our circuit it will be left out.

- `if (ADC10MEM < 0x266)  //Low REM Voltage (1.8V is where REM starts, REM is between 1.8 and 2.8V)`

The content of this "if" statement is the algorithm for detecting REM. ADC10MEM stores the digital value from the electrodes that the user is wearing (analog signal has been converted to digital, ADC10MEM is the digital value for the analog signal) and then ADC10MEM is compared to a chosen threshold (value). This value is set for a lower threshold, everything below this value in not considered as finding REM. This threshold is used to differentiate between reference noise and actual eye movements. The main goal of this threshold is to not include noise from the power grid or the circuit to be tracked as REM. To solve for the lower threshold see below:

$$\frac{\text{Desired threshold}}{\text{ADC Register Reference}} * (\text{ADC \# bits}) = \text{"Decimal value for ADC threshold"}$$

For an example I will use 1.8v as a desired threshold.
The ADC Register Reference is selected to use our battery as the reference voltage (as seen above in the register section), our battery is 3v to 0 (GND).
The MSP430G2553 has a 10 bit ADC

$$\frac{1.8\text{v}}{3\text{v}} * (2^{10}) = \text{"Decimal value for ADC threshold"}$$

$$0.6 * 1024 = \text{"Decimal value for ADC threshold"}$$

$$614.4 = \text{"Decimal value for ADC threshold"}$$

After this the decimal value 614 was converted to a hexadecimal value which equals 266. If ADC10MEM is less than 1.8 volts it will enter this "if" statement. During this statement an LED indicator is triggered to notify the user that they are not in REM.

```
{
        P1OUT &= ~0x01;                 //Clear P1.0 LED off
        P1OUT |= 0x10;                  //Set P1.4 LED on
}
```

A lower threshold at 1.8v was chosen because our signal is centered at 1.5v, setting 1.8 as the lower threshold helped cancel out any other noise that was not taken care of by our hardware filters.

❖ `else if (ADC10MEM > 0x3BB)  // High (over 2.8V is saturated)`

The next part of the "if" statement is the "else if" for the upper threshold. This is used to cancel out any eyebrow movements or other muscle movements not from the eyes. If large muscle movements occur the signal will saturate at 3 volts, these movements should not be counted as REM. To pick a proper threshold it is important to see the maximum voltage that REM reaches. After the max REM voltage is found everything above should not be considered as eye movements, so the high threshold can be chosen to be at or a little above the maximum REM voltage, this will differ between people (note that our hardware has a gain control to adjust the gain of the signal). We found that 2.8v was a proper high threshold for our testing. Using the method above, a chosen voltage at 2.8 makes a hexadecimal value of 3BB.

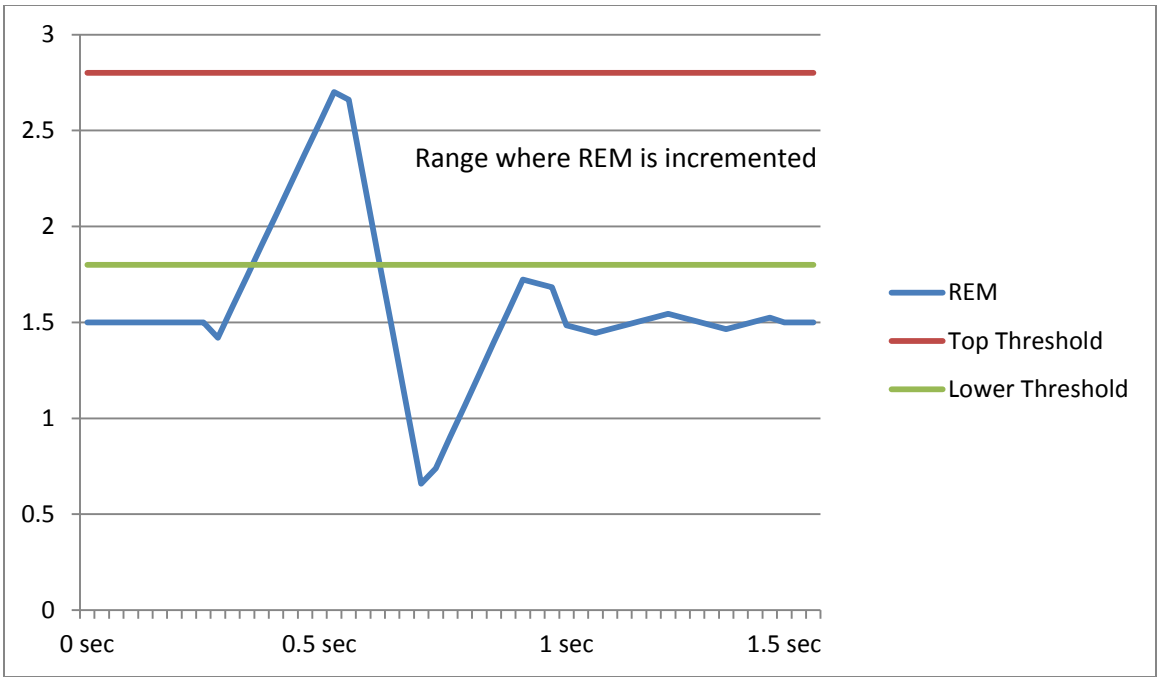
```
{
        P1OUT |= 0x10;      //Set P1.4 LED on
        P1OUT &= ~0x01;     //Clear P1.0 LED off
        REM--;              // Subtracts to REM if in voltage range
}
```

When the "if else" statement is entered, REM is decreased for the amount of time the signal is over the chosen high threshold. Anything inside the threshold increments the 'REM' variable. Anything over the upper threshold decrements the 'REM' variable. This is done to eliminate the REM detection during eye movements or large muscle movements. Also the "REM detection" LED is off and the "REM not detected" LED is on.

❖ `Else`

The last part of the REM detection code is an "else" statement that increments REM. In the above scenario, for signals below the lower threshold or when the signals are greater than the upper threshold, REM is not detected. All that is left is the scenario when the signal is within the upper and lower thresholds. In this case REM is detected and the "REM detection" LED is on and the "REM not detected" LED is off.

```
P1OUT |= 0x01;      //Set P1.0 LED on
P1OUT &= ~0x10;     //Clear P1.4 LED off
REM++;              // Adds to REM if in voltage range
```



What is expected when an eye movement is picked up by the REM Monitor II, Voltage with respect to time in seconds

Figure 23: REM Threshold

The "if" statements seen between the "REM increment" and PWM code are directly related to the set variables in the "Initialize Variables" section. Look at the above "Initialize Variables" to understand the algorithm between the REM detection algorithm and the Pulse Wave Modulation code.

Pulse Wave Modulation

Once the desired "NumberOfScanPeriods" is met, the "timercount" variable will be greater than 460, this will cause the code to stay out of all "if" statements related to "ScanPeriodInREM" and "NumberOfScanPeriods". ADC values are still taken in and sent via UART to the Bluetooth, and the REM indicator lights are still active. When the code enters the PWM stage, it will eventually require a hard reset.

```
❖ if((timerCount>465))      // For this statement timer is sped up, 1second = 500
  timerCount
          {
                P1DIR |= (1<<6);              //P1.6 output
                P1SEL |= (1<<6);              //P1.6 TA1/2 options
                CCTL1 = OUTMOD_6;           // CCR! toggle/set
                TACTL = TASSEL_2 + MC_3;   // SMCLK, up-down mode
```

P1.6 is used as an output for the speaker; the above lines initialize P1.6 for the speaker. CCTL1 then sets up P1.6 so that it can toggle to PWM and TACTL sets the clock, more details about this can be found in the MSP430 data sheet.

```
CCR0 = 1000;                    // PWM Period/2
```

$$\left(\frac{\text{CCR0 Value}}{4*\text{Clock Speed}}\right)^{-1} = \text{Frequency}$$

$$\left(\frac{1000}{4*1000000}\right)^{-1} = \text{Frequency}$$

$$(0.004)^{-1} = \text{Frequency}$$

$$250\text{ hz} = \text{Frequency}$$

The frequency can be controlled by setting CCR0. The CCR0 value is a multiple the period, therefore you must divide it by 4. This equation will work for picking a frequency.

The next change that can be made is with the duty cycle.

```
CCR1 = 5;                              // CCR1 PWM duty cycle
```

The duty cycle is easy to set, all that is required a percent on time, and that value can be set for CCR1.

$$\frac{\text{CCR1}}{100} = \text{Duty Cycle}$$

$$\frac{5}{100} = \text{Duty Cycle}$$

$$5\% = \text{Duty Cycle}$$

For this example the duty cycle is 5%, this means that the wave will be high 5% and low 95%, in other words, for five clock ticks the wave will be high and for 95 clock ticks the wave will be low. This can be used to adjust the volume for each individual user. We have set our duty cycle low because we do not want the user to be awakened when being cued while they are in REM.

The high frequency PWM is basically the same. The two changes are to the frequency and the duty cycle.

```
CCR0 = 100;             // PWM Period/2
CCR1 = 50;              // CCR1 PWM duty cycle
```

The same equations can be used as above to solve for these values, the frequency comes to being 2.5 Khz and the duty cycle is 50%. We chose a high frequency with a higher duty cycle to make sure the user would wake up from this tone.

Interrupts

The follow section will discuss the interrupts that are used and explain in a little more detail what is happening in the interrupts.

```
// Timer A0 interrupt service routine
❖ #pragma vector=TIMER0_A0_VECTOR
  __interrupt void Timer_A (void)
```

Setting up the TIMER0 interrupt can be found in the MSP430G2553 data sheet, the main thing to look at is what's happening inside this interrupt. There first thing that happens in this interrupt is timerCount is incremented

```
❖ timerCount = (timerCount + 1); // Clock at 1Mh (time per clock =65536/1000000)
```

The timerCount variable is used to keep track of time in an easy straight forward manner. We solved for how much timerCount equals earlier, by dividing the amount of data by the clock. Every timerCount value stands for a certain amount of time seen below.

$$(\text{timerCount}) * (0.065536) = \text{"Time in Seconds for timerCount"}$$

For example if timer count equals 457 how many seconds would timerCount be.

$$(457) * (0.065536) = \text{"Time in Seconds for timerCount"}$$

$$29.95 = \text{"Time in Seconds for timerCount"}$$

This is how the time periods are set to a chosen value, making the program very usable and easy to change time periods. The next statement in the TIMER0 interrupt is related to the ADC data being sent.

```
❖ if(timerCount%2)        // Send data every other timerCount, (too much data slows
  LabVIEW)
```

During our testing process we tried collecting and sending data in different places of the code and at different speeds. If the data was sent too fast LabVIEW (the program that was processing the data) would not show the data real time. We found that sending the data every other timerCount (about every .13 seconds) would keep LabVIEW running real time. The modulus is set to modulate timerCount by 2 making the "if" statement only entered on every other timerCount increment. Note that it is possible to send data faster, but for our demonstration purposes this was enough. When looking at an 8 hour period of data collection, this comes to be a large quantity of data even with the slower send rate.

```
❖ UCA0TXBUF = ADC10MEM;  // UART stored and sent in UCA0TXBUF
```

The only statement inside the "if" statement is the UART store and send line. ADC10MEM stores the ADC value and UCA0TXBUF transmits the data. For more information on this refer to the MSP430G2553 data sheet. The next statement is related to the Low Power Mode.

```
❖ if(timerCount>18280) // 18280 is approx 20 min, MSP430 is in sleep mode for 20
  min // Used for Low power mode, With ADC, interrupt, and no crystal Not Efficient
      {
            _BIC_SR_IRQ(LPM0_bits);                  // Clear LPM3 bits from 0(SR)
      }
```

As mentioned above at the start of the Main, the Low Power mode is an option. This is the exit service routine. The wait period can be set to whatever desired amount by setting the value in the "if" statement.

```
❖ // ADC10 interrupt service routine
  #pragma vector=ADC10_VECTOR
  __interrupt void ADC10_ISR(void)
     {
     __bic_SR_register_on_exit(CPUOFF);            // Clear CPUOFF bit from 0 (SR)
     // __delay_cycles(500); If adc acts to fast, delay can be used
     }
```

For the last part of the code the ADC10 interrupt is enabled. This sets up the ADC and makes and stores the digital value for the analog point in ADC10MEM. The only other line not strictly related to the ADC is the __delay_cycles(500) this is used to slow the ADC down in case it reads or transmits to fast. This problem is solved in the above timer interrupt. For more information about the ADC10 interrupt service routine refer to the MSP430G2553 data sheet.

Code Summary

The last few pages described in detail some of the more complicated parts of the code. The software used for programming the MSP430 is Code Composer Studio 5.1. The software is free and there are many tutorials available online for starting a MSP430 project.

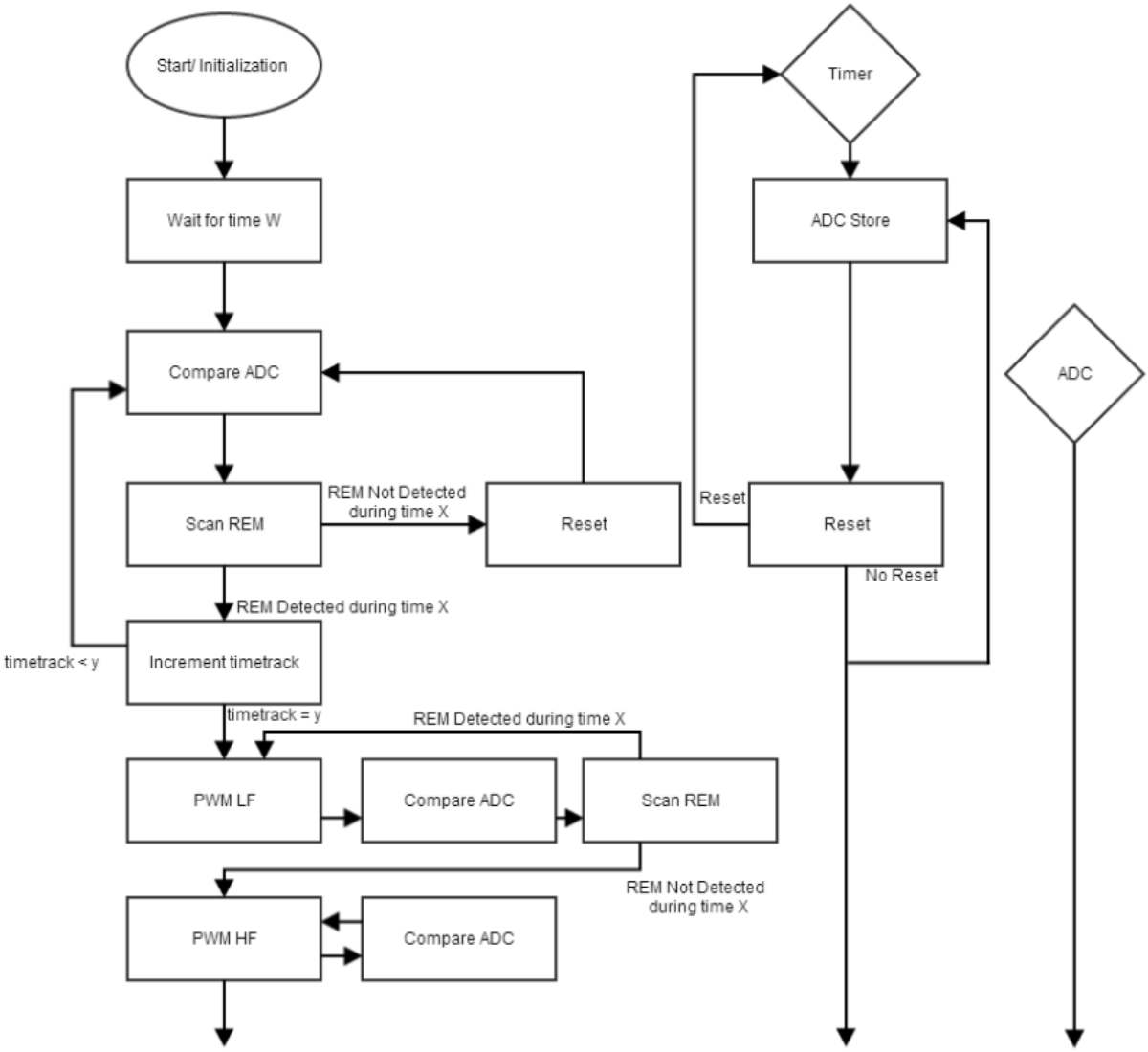


Figure 24: Software Flow Chart